Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 1999 Proceedings

Americas Conference on Information Systems (AMCIS)

December 1999

A Distributed Architecture to Customize CASE Tool Prototypes

Lei Jin
Georgia State University

A Distributed Architecture to Customize CASE Tool Prototypes

Lei Jin
Computer Information Systems Department
Georgia State University
Atlanta, GA 30302
e-mail: jinlei@gsu.edu

Introduction

A CASE tool is a computer-based software that supports one or multiple system development methods (Jarzabek and Huang, 1998) at different stages of system development. It is primarily introduced to help system analysts and developers to build system more consistently and efficiently. It is noted that most of the system development methodologies, including Object Oriented modeling methodology, are deeply embedded in the CASE tools (Adhikari, 1996). A survey of the CASE tool market showed that the annual world wide market for CASE tools was $4.8 billion in 1990 and grew to approximately $12.11 billion in 1995 (Jarzabek and Huang, 1998).

On the other hand, some evidence indicates that CASE tools are "dearly bought but sparsely used". Jarzabek and Huang argue that software development should be viewed as a creative, problem-solving process rather than a restrictive method employment practice. That is why purely method-centered CASE tools can not fulfill the important needs of software developers, especially, the "soft" aspect of software development. This "soft" aspect of software development is defined as the problem solving elements related to system design, such as creativity, understanding, idea generation, experiment with alternative solutions, etc. Most of the current CASE tools are primarily method driven, providing very limited support on problem-solving aspects of the software development (Jarzabek and Huang, 1998).

This paper proposes a distributed architecture to allow software developers to customize or build CASE tool prototypes according to their own preferences. Even though this approach doesn't cure the problem pointed out by Jarzabek and Huang, I do believe it makes an important step toward user-centered CASE tool design, especially CASE tool interface design. And both software developers and CASE tool vendors can potentially benefit from the proposed architecture.

It's possible for developers to "experiment" with and configure the CASE tools according to specific system development project requirement. For example, based on the project team members' past experience with system development method and CASE tool, an initial set of choices on CASE tool supporting features can be made and required from the CASE tool vendor. After the vender assembled those components and install them directly to the developer's desktop, software developers can start to use the tool to perform various system development related activities within a specific development project context. During this process, the developers can flexibly acquire new features, modify their original configurations, modify the notations, diagrams or reports they would like the CASE tool to support, or even come up with their own notifications, diagrams and reports and require them to be incorporated in the CASE tool. Based on the feedback from developers, the vendor can assemble a new package of CASE tool environment and install them for the developers. Notice that this is an ongoing process, software developers and CASE tool vendors work together, continuously explore and discover the best fit between a CASE tool and a specific development task.

This approach can also be beneficial to the CASE tool vendors. The frequent feedback from software developers can potentially help vendors improve the quality of CASE tool design. The dynamic assembly capability may also bring CASE tool vendors new crowd of customers. For example, vendors can easily scale down a sophisticated CASE tool environment to some basic features and sell them to students or other end users.

Review of Enabling Technologies

Three requirements are necessary to build CASE tool prototyping architecture:

1. The capability of mass-customization;
2. The ability to dynamically assemble the CASE tool components at real time;
3. The mechanism to deliver the prototype conveniently and safely to customers.

Internet as Primary Distribution Channel: Given that more and more organizations link their corporate networks to the Internet, it can be utilized as the primary channel to deliver the CASE tool prototypes. Moreover, customers can easily go to vendor's website to specify

their preferences and modify their configurations regarding CASE tool prototypes.

CORBA and Distributed Object Computing: According to Mowbray et al. (1997) and Hart et al. (1995), CORBA (Common Object Request Broker architecture) is a leading standard for DOM (Distributed Object Management). It combines Object Oriented technology with a client-server model to provide a uniform view of an enterprise's computing system in terms of: (1) Uniform access to services; (2) Uniform discovery of resources and object names; (3) Uniform error handling methods; (4) Uniform security policies. If we can model the CASE tool as a distributed and complex object class with different components, the features that CORBA supports will enable dynamic assembling of CASE tool prototypes from various CASE component objects that may reside in different network nodes.

CORBA are operated through "Object Request Broker" (ORB), which can be viewed as a peer-to-peer communication layer that resides in both client and server. ORB provides a common object-oriented, remote procedure call mechanism that allows dynamic binding of objects and services. The CORBA object model is illustrated in Figure 1.

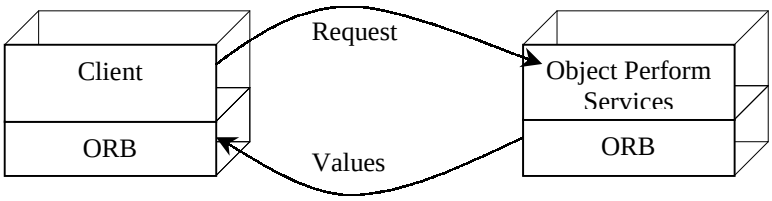

Figure 1: CORBA Object Model

Castanet and Push Technology

Marimba's Castanet system is a push technology that provides three core components—the Publisher, Transmitter, and Tuner—to facilitate the publishing, distributing and managing Castanet Channels. Channels are executable applications or data files to be distributed from server to client across any TCP/IP network. After Channels are created, the Castanet Publisher will automatically transfer all of the files to the corresponding primary Transmitters as well as repeating Transmitters. The Channel subscribers can run Castanet Tuner to subscribe the Channels. The Castanet Tuner not only download Channel content, but also be able to install it, update it automatically as each new version of the Channel is released, and remove files that are superseded. Castanet also features advanced security protection mechanisms such as VeriSign certificate verification, SSL and access control. More detailed information about the Castanet system is available from their website: http://www.marimba.com.

Architecture for CASE Prototyping

The architecture that potentially enables CASE tool prototyping and customization will incorporate all three key technologies components that are previously reviewed. As indicated in Figure 3, at the CASE tool vender site, a CORBA architecture is implemented to ensure the distributed object dynamic binding, this requires that each network node interfaced with ORB. Inside Lan1, the "Method Integration Component" is connected with various method component servers, including Requirement definition model component, Analysis model component, Design model component, and implementation component. The data dictionary component and report generation component are also involved. These components are derived from Yourdon and Argila's (1996, p.3) the "generic model-based software engineering life cycle". Even though object oriented methods and traditional system development methods are different in many perspectives, they can be decomposed into parallel components because "virtually all modern software engineering approaches have adopted this model-based 'philosophy.' What varies greatly from one software engineering method to another is the kinds of models that should be built, how they should be built and who should built them." (Yourdon and Argila, 1996)

The Method Integration Component will be responsible for requesting all of the object components that are relevant to a particular system development method that one customer requires. It not only knows where to get which components, but also conducts the consistency checking to see if the principle, criteria and guideline (Song, 1997) regarding the method are violated. After all of the consistent components are obtained, they will be integrated into one method component, which will be further sent to the "CASE Tool Prototype Integration Component", which will further incorporate GUI component, online reference component and other generic components into the CASE tool prototype. This customized prototype will be launched as a channel through Castanet Publisher, and be delivered and installed on the customer's computer through Castanet Transmitter and Tuner. Whenever the customer wants to modify the CASE tool, he/she can go to vender's website to change

the configuration, and the correspondingly reassembled CASE tool prototype will be integrated, launched and installed.

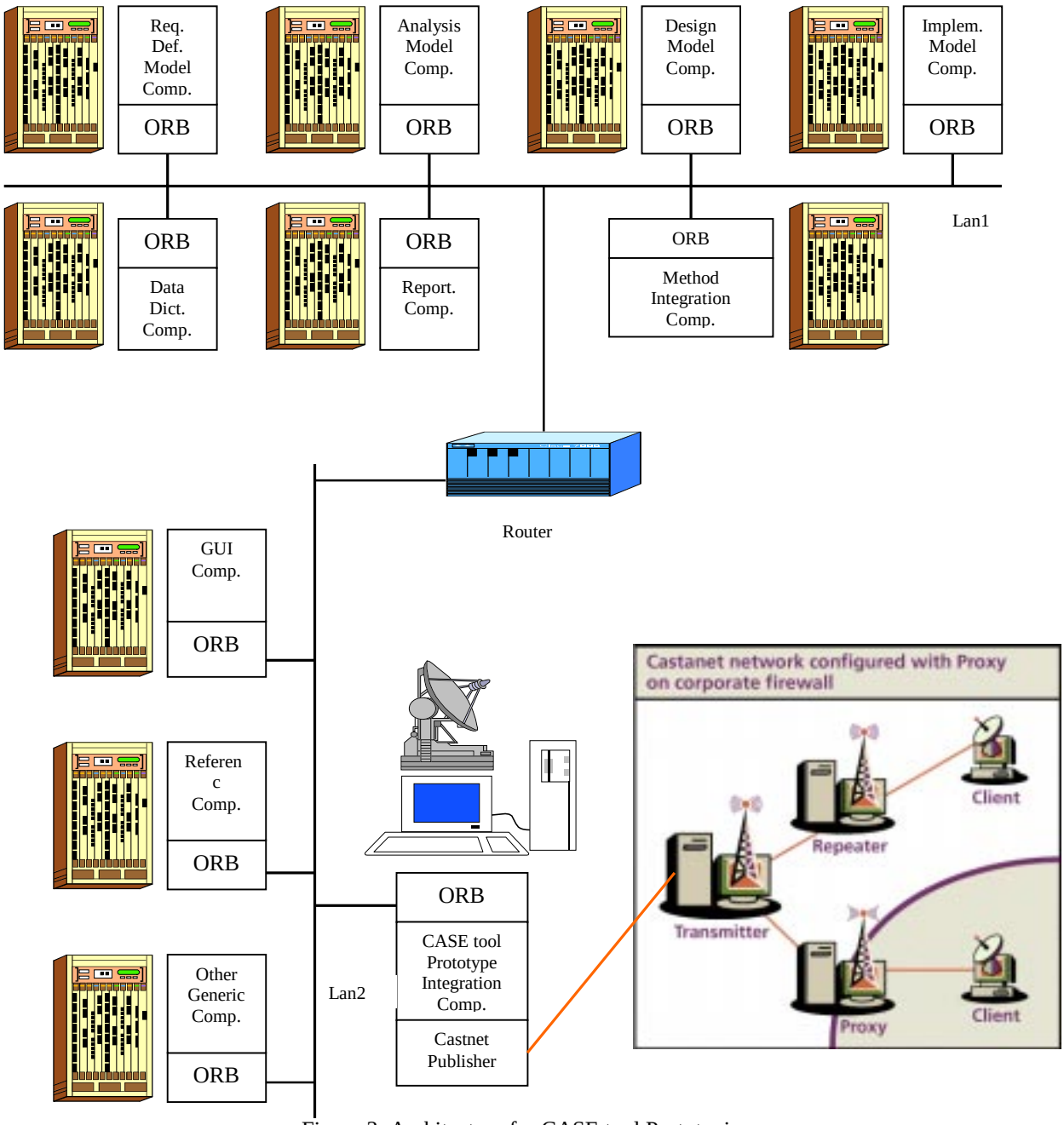

Figure 3: Architecture for CASE tool Prototyping

Summary and Future Work

This paper examines the CASE tool design from object-oriented modeling and prototyping perspectives. The proposed architecture leverage advanced information technologies—including distributed object computing and push technology—to enable CASE tool dynamic customization and prototyping capabilities. The implementation of the architecture can potentially benefit CASE tool venders, system analysts and developers who use CASE tool to design systems, as well as students who learn to use CASE tools. The future work intends to explore the detailed specification of the object components of CASE tools and provide examples to illustrate the decomposition process of the CASE tool components